CLAIMS

1.   A computing device programmed with an extensible framework that accepts one or more mark-up language parsers and/or generators, each implemented as plug-ins to the framework, with different plug-ins enabling different kinds of mark up languages to be handled by the device.

2.   The computing device of Claim 1 in which the extensible framework (i) insulates a client running on the device from having to communicate directly with a parser or generator and (ii) is generic in that it presents a common API to the client irrespective of the specific kind of parser or generator deployed.

3.   The computing device of Claim 1 or 2 in which the client interacts with several kinds of parser or generator plug-ins to the extensible framework, each handling different mark-up language formats.

4.   The computing device of any preceding Claim programmed with a file conversion capability requiring (i) a source file to be parsed by the parser, which is adapted to handle one format and (ii) an output, converted file to be generated by the generator, adapted to handle a different file format.

5.   The computing device of any preceding Claim in which several different clients are able to share the same parsers or generators.

6.   The computing device of any preceding Claim in which the extensible framework enables a parser or generator to access data from any source that conforms to a generic data supplier API.

7.   The computing device of Claim 6 in which the source is a buffer in memory.

8.   The computing device of Claim 6 in which the source is a file.

9.   The computing device of Claim 6 in which the source is a socket outputting streaming data.

10. The computing device of any preceding claim 6 - 9 which uses a data source different from any data sources which the device was capable of using when first operated by an end-user.

11. The computing device of any preceding Claim, in which the extensible framework enables the mark-up language parser or generator to access components to validate, pre-filter or alter data, in which the components are plug-in components to the framework and operate using a chain of responsibility design pattern.

12. The computing device of Claim 11 in which the plug-in components all present a common, generic API to the parser or generator, enabling the same plug-in components to be used with different types of parsers and generators.

13. The computing device of Claim 11 in which the plug-in components all present a common, generic API to a client component using the parser or generator, enabling the same plug-in components to be used by different clients.

14. The computing device of any preceding claim 11- 13 in which the parser notifies a validator plug-in of elements it is parsing and these in turn go to an auto correction plug-in to be fixed if required and finally a client receives these events.

15. The computing device of any preceding claim 11- 14 in which a parsed element stack is made available to all validation/pre-filter/altering plug-ins.

16. The computing device of any preceding claim 11- 15 which incorporates a character conversion module that enables documents written in different character sets to be parsed and converted to a common, generic character set.

17. The computing device of any preceding claim in which extensions to its capabilities can be made without affecting compatibility with existing clients or existing parsers and generators by the use of an updated/extended namespace plug-in that sets-up all the elements, attributes and attribute values for a namespace.

18.   A method of parsing a mark-up language document, comprising the step of accessing an extensible framework that accepts parser plug-ins, with different plug-ins enabling different kinds of mark up languages to be handled.

19.   A method of generating a mark-up language document, comprising the step of accessing an extensible framework that accepts generator plug-ins, with different plug-ins enabling different kinds of mark up languages to be handled.

20.   The method of Claim 18 or 19, in which extensions to device capabilities can be made without affecting compatibility with existing clients or existing parsers and generators by the use of an updated/extended namespace plug-in that sets-up all the elements, attributes and attribute values for a namespace.

21.   The method of any preceding method Claim, in which the extensible framework (i) insulates a client running on the device from having to communicate directly with a parser or generator and (ii) is generic in that it presents a common API to the client irrespective of the specific kind of parser or generator deployed.

22.   The method of any preceding method Claim, in which the client interacts with several kinds of parser or generator plug-ins to the extensible framework, each handling different mark-up language formats.

23.   The method of any preceding method Claim, in which the device is programmed with a file conversion capability requiring (i) a source file to be parsed by the parser, which is adapted to handle one format and (ii) an output, converted file to be generated by the generator, adapted to handle a different file format.

24.   The method of any preceding method Claim, in which several different clients are able to share the same parsers or generators.

25.   The method of any preceding method Claim, in which the extensible framework enables a parser or generator to access data from any source that conforms to a generic data supplier API.

26.   The method of preceding Claim 25, in which the source is a buffer in memory.

27.   The method of preceding Claim 25 in which the source is a file.

28.   The method of preceding Claim 25 in which the source is a socket outputting streaming data.

5

29.   The method of any preceding Claim 25 – 28, in which a data source is used that is different from any data sources which the device was capable of using when first operated by an end-user.

30.   The method of any preceding method Claim, in which the extensible framework

10   enables the mark-up language parser or generator to access components to validate, pre-filter or alter data, in which the components are plug-in components to the framework and operate using a chain of responsibility design pattern.

31.   The method of preceding Claim 30, in which the plug-in components all present a common, generic API to the parser or generator, enabling the same plug-in

15   components to be used with different types of parsers and generators.

32.   The method of any preceding Claim 30 – 31, in which the plug-in components all present a common, generic API to a client component using the parser or generator, enabling the same plug-in components to be used by different clients.

33.   The method of any preceding Claim 30 – 33, in which the parser notifies a

20   validator plug-in of elements it is parsing and these in turn go to an auto correction plug-in to be fixed if required and finally a client receives these events.

34.   The method of preceding Claim 30 – 33, in which a parsed element stack is made available to all validation/pre-filter/altering plug-ins.

25

35.   The method of any preceding Claim 30 – 34, in which a character conversion module enables documents written in different character sets to be parsed and converted to a common, generic character set.